AMENDMENTS TO THE CLAIMS

The following is a complete, marked-up listing of revised claims with a status identifier in parenthesis, underlined text indicating insertions, and strike through and/or double-bracketed text indicating deletions.

LISTING OF CLAIMS

1. (Previously Presented) A method for modifying software, comprising:

initially forming, from an original piece of software including only source text, a hybrid form of the original software, formed in such a way that at least one part of the source text is compiled into at least one of a byte and binary code and at least one further part of the source text is converted into a code formulated in a meta markup language for at least one variation point;

subsequently converting only at least one variation point of the hybrid form of the original software as necessary by a transformation in accordance with transformation rules into at least one other code formulated in the meta markup language; and

forming a modified variation point of an adapted piece of at least one of software and a source code from said other code via a converter and then forming at least one of a binary and byte code of the modified variation point of an adapted piece of software via a compiler, the original and the adapted software differing in terms of at least one of their program execution and program content.

2. (Previously Presented) The method as claimed in claim 1, wherein the transformation rules have at least one modification rule for a variation point.

3. (Previously Presented) The method as claimed in claim 1,

wherein the modification rule initiates an update to at least one of a more recent software version or a patching operation.

4. (Previously Presented) The method as claimed in claim 1,

wherein the modification of at least one variation point is performed by way of the transformation at runtime.

5. (Previously Presented) The method as claimed in claim 1,

wherein the programming language of the source code is Java and the meta markup language of the variation points is XML and wherein the transformation and the rule description are implemented via XSLT and XSL.

6. (Previously Presented) A method for modifying source code, comprising:

making a first code formulated in a meta markup language with language extensions formulated in at least one meta markup language available as the source code;

converting the source code, via a transformation in accordance with transformation rules, into a second code formulated in the meta markup language without the language extensions formulated in the meta markup language;

using the transformation rules to form a language converter which at least one of resolves and applies the language extensions of the first code in such a way that they can be processed by a back-converter that has no corresponding language extension; and

converting said second code into a second source code that is formulated in at least one of the first programming language and a different programming language and yields at least one of a valid binary code and byte code.

7. (Previously Presented) The method as claimed in claim 6,

wherein at least one language extension is at least one of newly generated in the second code and taken over from the first code, and this at least one of generation and takeover is performed by the language converter.

8. (Previously Presented) A method for modifying source code, comprising:

converting a source code, formulated in a first programming language, into a first code formulated in a meta markup language;

transforming the first code, exclusively in accordance with transformation rules, into a second code formulated in the meta markup language; and

transforming the second code into a second source code formulated in at least one of the first programming language and a different programming language, the first and the second source code differing in terms of their functionality.

9. (Previously Presented) The method as claimed in claim 8,

wherein the transformation rules include at least one condition and at least one of one logic component and code fragment itself.

10. (Previously Presented) The method as claimed in claim 8,

wherein transformation rules include at least one fragment in the form of at least one of a template and at least one pattern in which at least one code modification is effected with the aid of the transformation.

11. (Previously Presented) The method as claimed in claim 10,

wherein the transformation rules are embodied in such a way that a mechanism for backing up at least one system state is incorporated into the second source code with the aid of the transformation in order to enable a migration into other versions.

12. (Previously Presented) The method as claimed in claim 8,

wherein at least one template is formed from at least one of the first code and a fragment of the first code with the aid of the transformation.

13. (Previously Presented) A method for modifying source code, comprising:

converting a source code, formulated in a first programming language, into a first code formulated in a meta markup language;

adding an item of information formulated in the meta markup language and influencing the subsequent program execution, via a transformation, to the first code in at least one of a substituting and non-substituting way and wherein in this way, a second code also formulated in the meta markup language is formed, the transformation being performed in accordance with transformation rules formulated in a transformation description language; and

transforming the second code into a second source code formulated in at least one of the first programming language and a different programming language, at least one of the program content and program execution of the first source code differing from at least one of the program content and program execution of the second source code.

14. (Previously Presented) The method as claimed in claim 13, wherein said information includes at least one code fragment and wherein the second source code is formed in that at least one code fragment contained in the first source code is replaced with the aid of the transformation by the at least one code fragment contained in the fragment.

15. (Previously Presented) The method as claimed in claim 13,

wherein the information specifically includes data in the form of at least one of initialization states, state data and database data.

16. (Previously Presented) The method as claimed in claim 15,

wherein the transformation rules are influenced by the data.

17. (Previously Presented) The method as claimed in claim 13,

wherein at least one of the data and code fragments are additionally embedded in the transformation rules.

18. (Previously Presented) The method as claimed in claim 13,

wherein data generated by checkpoints is added by a transformation in such a way that the internal state of the original program is at least one of available at the restart of the program and is usable by the program.

19. (Previously Presented) The method as claimed in claim 13,

wherein information includes at least one of updates and patches.

20. (Previously Presented) The method as claimed in claim 13,

wherein fragments contain internationalization information which serves for the adaptation to different natural languages.

21. (Previously Presented) The method as claimed in claim 13,

wherein at least one of data and code fragments originate from a library and carry information tailored to at least one of customers and customer groups.

22. (Previously Presented) A non-transitory computer readable medium encoded with computer executable instructions for modifying source code, the set of computer executable instructions comprising:

presenting a hybrid form of an original piece of software in such a way that at least one part of a source text is compiled into at least one of a byte and binary code and at least one further part of the source text is converted into a code formulated in a meta markup language for at least one variation point;

presenting a device for transformation in such a way that only at least one variation point of the hybrid form of the original software is convertable as necessary via the transformation in accordance with transformation rules into at least one other code formulated in the meta markup language,

whereby said other code directly forms a modified variation point of at least one of an adapted piece of software and a source code is formable from the other code by a converter and then at least one of a binary and byte code of the modified variation point of an adapted piece of software is formable by a compiler and whereby the original and the adapted software differ in terms of at least one of their program execution and program content.

23. (Previously Presented) A non-transitory computer readable medium encoded with computer executable instructions for modifying source code, the set of computer executable instructions comprising at least one of:

presenting a processor in such a way that a transformation of the source code is performed in accordance with transformation rules formulated in an extended style description language and a language converter contained therein in such a way that either a second code formulated in the meta markup language without the language extensions of the first code that were formulated in the meta markup language; and

generating a second code formulated in the meta markup language using at least one of new and the original language extensions formulated in the meta markup language, wherein a back-converter is present in such a way that the second code is transformed into a source code formulated in at least one of the first programming language and a different programming language.

24. (Previously Presented) A non-transitory computer readable medium encoded with computer executable instructions for modifying source code, the set of computer executable instructions comprising:

presenting a first converter in such a way that a source code formulated in a first programming language is converted into a first code formulated in a meta markup language;

presenting a processor in such a way that the first code is converted via a transformation exclusively in accordance with transformation rules into a second code formulated in the meta markup language; and

presenting a second converter in such a way that the second code is converted into a second source code formulated in at least one of the first programming language

and a different programming language, the first and the second source code differing in terms of at least one of their functionality and content.

25. (Previously Presented) A non-transitory computer readable medium encoded with computer executable instructions for modifying source code, the set of computer executable instructions comprising:

presenting a first converter in such a way that a source code formulated in a first programming language is converted into a first code formulated in a meta markup language;

presenting a processor in such a way that an item of information formulated in the meta markup language and influencing the subsequent program execution is added by a transformation to the first code in at least one of a substituting and non-substituting manner and in this way the second code also formulated in the meta markup language is formed, the transformation being performed in accordance with transformation rules formulated in a transformation description language; and

presenting a second converter in such a way that the second code is transformed into a second source code formulated in at least one of the first programming language and a different programming language, at least one of the program content and program execution of the first source code differing from at least one of the program content and program execution of the second source code.

26. (Previously Presented) The method as claimed in claim 6,

wherein the modification rule initiates an update to at least one of a more recent software version or a patching operation.

27. (Previously Presented) The method as claimed in claim 6,

wherein the modification of at least one variation point is performed by way of the transformation at runtime.

28. (Previously Presented) The method as claimed in claim 9,

wherein transformation rules include at least one fragment in the form of at least one of a template and at least one pattern in which at least one code modification is effected with the aid of the transformation.

29. (Previously Presented) The method as claimed in claim 6,

wherein at least one template is formed from at least one of the first code and a fragment of the first code with the aid of the transformation.

30. (New) The method as in claim 1, wherein the variation point corresponds to an update to the source code.

31. (New) The method as in claim 1, wherein the variation point corresponds to a patch of the source code.

32. (New) The method as of claim 1, wherein the variation point corresponds to a user-specific requirement.

*** END CLAIM LISTING ***